Appeal No. 2017-1118

IN THE UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

ORACLE AMERICA, INC.,

Plaintiff-Appellant,

v.

GOOGLE INC.,

Defendant-Cross Appellant.

Appeal from the United States District Court for the Northern District of California in Case No. 10-CV-3561, Judge William H. Alsup

BRIEF OF AMICI CURIAE EUGENE H. SPAFFORD, PH.D., ZHI DING, PH.D., ADAM PORTER, PH.D., AND KEN CASTLEMAN, PH.D., IN SUPPORT OF APPELLANT

Jared Bobrow
Principal Attorney
Aaron Y. Huang
Amanda K. Branch
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

February 17, 2017

Counsel for Amici Curiae

TABLE OF CONTENTS

TABLE OF AUTHORITIES

CERTIFICATE OF INTEREST

Counsel for amici curiae, Eugene H. Spafford, Ph.D., Zhi Ding, Ph.D., Adam Porter, Ph.D., and Ken Castleman, Ph.D., certifies the following:

1. The full name of every party represented by me is:

EUGENE H. SPAFFORD
ZHI DING
ADAM PORTER
KEN CASTLEMAN

2. The names of the real parties in interest (if the party named in the caption is not the real party in interest) represented by me is:

N/A

3. All parent corporations and any publicly held companies that own 10% or more of the stock of the party represented by me are:

N/A

4. The names of all law firms and the partners or associates that appeared for the parties now represented by me in the trial court or agency or are expected to appear in this Court are:

Jared Bobrow
Aaron Y. Huang
Amanda K. Branch
WEIL, GOTSHAL & MANGES LLP

Dated: February 17, 2017

/s/ Jared Bobrow
Jared Bobrow
WEIL GOTSHAL & MANGES LLP

I.

STATEMENT OF INTEREST

Dr. Eugene H. Spafford is a Professor of Computer Science at Purdue University, where he has been employed since 1987, and the founder and Executive Director of the Center for Education and Research in Information Assurance and Security at Purdue. He has over 30 years of experience in both practice and research in the field of computing and computer science, including with the use of application programming interfaces ("APIs"). Over the past decade, he has served in an advisory or consulting capacity on issues in computing and information systems with several U.S. government agencies and their contractors, including the National Science Foundation, the Federal Bureau of Investigation, the Government Accountability Office, the National Security Agency, the U.S. Department of Justice, the U.S. Air Force, the U.S. Naval Academy, U.S. SPACECOM, the Department of Energy, and the Executive Office of the President. He has served on the President's Information Technology Advisory Committee and has testified before Congressional committees nine times. He is a Fellow of five major scientific and professional organizations involved with computing, the Association for Computing Machinery, American Academy for the Advancement of Science (AAAS), Institute of Electrical and

Electronic Engineers ("IEEE"), International Information Systems Security Certifications Consortium, and Information Systems Security Association.

Dr. Spafford has published and spoken extensively about software engineering, information security, and professional ethics, and he has served on the editorial boards of several major journals of computer science. He was affiliated with the Software Engineering Research Center, an NSF University-Industry Cooperative Research Center located at Purdue. His current research is directed towards the architecture, construction, and public policy of secure information systems. He has been writing computer programs since 1972, including computer security programs that have been used internationally by government agencies and companies, and his programming experience includes Java and other programming languages. He also has taught undergraduate and graduate courses involving software engineering, information system security, and many programming languages.

Dr. Zhi Ding is a Professor of Electrical and Computer Engineering at the University of California, Davis. He has over 26 years of practical and research experience in the field of electronic and electrical engineering, including with the use of APIs. He received his Ph.D. degree in Electrical Engineering from Cornell University in 1990. He was a faculty member of Auburn University and the University of Iowa, and he has held visiting positions at Australian National

University, Hong Kong University of Science and Technology, NASA Research Center (Cleveland, Ohio), and USAF Wright Laboratory.

Dr. Ding has published extensively about electrical engineering, and his research focuses on communications and systems. Dr. Ding is a coauthor of the popular engineering textbook Modern Digital and Analog Communication Systems (4th ed.). Dr. Ding is a Fellow of the IEEE, and he has served on the technical committees of several workshops and conferences. He was the Technical Program Chair of the 2006 IEEE Global Telecommunication Conference. Dr. Ding's research includes active collaborations with researchers from several countries, including Australia, China, Japan, Canada, Taiwan, Korea, Singapore, and Hong Kong.

Dr. Adam Porter is a Professor of Software Engineering at the University of Maryland and the University of Maryland Institute for Advanced Computing Studies. Dr. Porter was appointed as the Executive Director of the Fraunhofer Center for Experimental Software Engineering in July 2015. The Fraunhofer Center is a UMD-affiliated applied research center focusing on software that increasingly underlies most innovation. Dr. Porter earned his Ph.D. from the University of California at Irvine.

Dr. Porter is a Senior Member of the IEEE and a Senior Member of ACM, and served on the editorial boards of the IEEE Transactions on Software

Engineering and the ACM Transactions on Software Engineering and Methodology. Dr. Porter has published extensively about software engineering, including the paper, Empirically Guided Software Development Using Metric-Based Classification, which was listed as one of the 20 most widely-cited articles published by IEEE Software. Dr. Porter has also taught a number of courses in software engineering, including in large scale software development. Dr. Porter also co-created a Massive Open Online course on Android development that has reached over 900,000 students.

Dr. Kenneth R. Castleman is the author of the highly regarded textbook, Digital Image Processing and co-editor of Microscope Image Processing. Dr. Castleman received his Ph.D. in Electrical Engineering from the University of Texas at Austin in 1970, and has over 45 years of experience in the field. Dr. Castleman is a former Adjunct Professor at the University of Texas at Austin, as well as former Senior Scientist at NASA Jet Propulsion Laboratory. Dr. Castleman also served as president of Advanced Digital Imaging Research, LLC, a leader in digital image processing research and development. Under his leadership, ADIR, and its predecessor, Perceptive Scientific Instruments, Inc., developed and deployed many innovations in image processing software under government funding.

Dr. Castleman has authored or co-authored more than 60 papers on digital image processing. Further, Dr. Castleman holds a number of image processing and image analysis patents. Dr. Castleman has served on many university and government advisory committees. For example, Dr. Castleman was a member of the Scientific Working Group on Imaging Technology for the Federal Bureau of Investigation, and also assisted NASA in the investigations of the Space Shuttle Challenger and Columbia disasters. He is also a Fellow of the American Institute of Medical and Biological Engineering and a member of the Space Technology Hall of Fame. Dr. Castleman has also offered his expertise in more than thirty intellectual property litigations since 1982.

Drs. Spafford, Ding, Porter, and Castleman's interest in this appeal is to improve the intellectual property laws of the United States. A robust and balanced intellectual property regime promotes innovation, reliability, and security in software and information systems. They have no interest in any party to this litigation or stake in the outcome of this appeal.

Drs. Spafford, Ding, Porter, and Castleman submit this *amici curiae* brief with the consent of all parties, who have stipulated their consent for all *amici* to file their briefs.

No party's counsel authored this brief in whole or in part. No party or party's counsel contributed money that was intended to fund preparing or submitting this brief. No person, other than the amici or their counsel, contributed money that was intended to fund preparing or submitting this brief.

II.

SUMMARY OF THE ARGUMENT

At stake in this appeal is whether Google's copying of Oracle's software code—specifically, certain packages of application programming interfaces ("APIs") for the Java programming platform—was fair use. We believe that there are important technical considerations that should be brought to bear on this determination.

As professors, researchers, and practitioners in computer science, we have created, used, and taught others about software APIs—including in the software that we have written, the research that we have overseen, the companies and government agencies that we have advised, and the courses that we have taught. Software APIs are widely used throughout modern, complex information systems. The public policy and legal treatment of APIs, as well as what constitutes fair use of those APIs, is therefore of great academic and practical interest to us and others in the computer science, computer engineering, systems engineering, electrical engineering, and software engineering communities.

We do not address the district court's legal reasoning, but we do observe that its holding appears predicated on certain assumptions regarding the creativity involved in an API, the effect of the platform on which an API is implemented on the purpose and expression of an API, and the need to copy an API from a technical perspective. Those assumptions, however, are misguided from our perspective as professors, researchers, and practitioners in the field of computer science.

This Court previously determined that the declaring code of the Java APIs has sufficient expression to be protectable by copyright. The creativity involved in Java APIs is similarly relevant to the second factor of the fair use determination, "the nature of the copyrighted work." The design and expression of an API reflects the creative choices and decision-making of its author. APIs can be expressed in many different ways yet still accomplish the same purpose and objective. Differences among APIs are therefore often due to creativity based on experience and experimentation, rather than dictates of functionality. The specific Java API packages at issue in this case provide a good example of the creative expression involved in designing an API: authoring the design of a Java API package requires significant creativity and involves many subjective choices not dictated by function—more than are required for authoring the code that implements that design.

Regardless of the medium on which the Java platform APIs are implemented, the expression and purpose of the APIs are the same. An API is not "transformed" merely because it was first implemented for use on desktop computers and later reimplemented on mobile devices. The meaning and message of the API are, from a technical perspective, identical, as the hardware form factors are simply computers.

Nor was it necessary for Google to copy the Java APIs. Google's copying of the Java APIs was not dictated by technical necessity, as Google did not achieve interoperability between Android and Java.

Because of the central role that well-designed APIs play in modern information systems, it is important that the law recognize and protect the creativity they embody.

III.

ARGUMENT

A. Technology Background

1. APIs and Software APIs

To assist the Court in understanding the technology of the present case, we provide the following discussion of APIs generally and of software APIs.

Generally speaking, an "application programming interface" is a designed expression that can be used to assist with software development. APIs come in a variety of types and forms, but all are used to describe and enable interactions in various types of applications. APIs can vary greatly in size, complexity, and form.

A “software API” is a type of API used in computing. A software API prescribes the expected behavior or set of rules that the software embodies, as well as the format and nature of data communicated with that software in any interaction. For example, a software API may describe what routines, data, and variables are available in the software; it may also describe the inputs, outputs, exceptions, and types of data used by the routines. A software API may also express all sorts of interrelationships with other APIs.

An API can consist of two different types of source code: a design that sets forth the structure and behavior of the API, and an implementation that accomplishes the function prescribed by that design. A single software API may have one or more implementations, in the form of one or more software libraries that each implement the same function called for by the design of the API. An API may also have no implementation, in the case of an abstract API used in design or documentation.

By analogy, an API is similar to a blueprint. The blueprint may express the design of a structure, and may specify the building materials, clearances and dimensions, placement of utilities, and methods of ingress and egress. The blueprint will incorporate features designed to meet the requirements of building codes, the strengths of the materials, and the capacities of other items not explicitly listed on the blueprint, such as wiring chases that may be used for a security system. There are countless designs that are possible because architects may make many subjective choices to address varying goals. The blueprint is therefore the creative expression of a unique design that is the product of the architect's many choices.

2. Java APIs

A software API can be for an application programming platform, in this case "Java." Java allows a user to organize functions and data into a hierarchy of nested and interrelated structures called "objects."[1] A Java API expresses the design of the objects, their relationships to each other, and prescribes the methods by which a program written in the Java language may interact with and control those objects. To accomplish this, a Java API expresses a set of "classes." A Java API organizes the classes into "packages," and the packages are organized into "libraries."

A Java API's definition for each class prescribes a set of controls and data, such as "variables" and "methods," associated with that class. A Java API may in turn define each method to have different inputs, known as "parameters"; outputs, known as "return values"; and values returned and/or actions taken in cases of error, known as "exceptions." These definitions of the parameters, return values, and exceptions in a Java API may be referred to as the "declaring code" for a

[1] There are several general forms of computer languages, and not all of them use the "objects," "classes," and "methods" described here as abstract representations for structure and definition. Those that do use these abstractions are known as "object-oriented" languages. Java is one such object-oriented language.

method. A program written in the Java language may call for the execution of a method defined by a Java API. The code which is interpreted and executed in order to execute the method may be referred to as the "implementing code." The implementing code is thus the executable code which, when executed, performs the function prescribed by a Java API, including interpreting the parameters and generating the return values and exceptions defined by the declaring code.

B. Technical Considerations Relevant To The Fair Use Factors

We understand that the considerations for a fair use inquiry are set forth in 17 U.S.C. § 107. We believe a number of technical considerations are relevant to the Court's application of these fair use factors to Google's copying of Oracle's Java API packages.

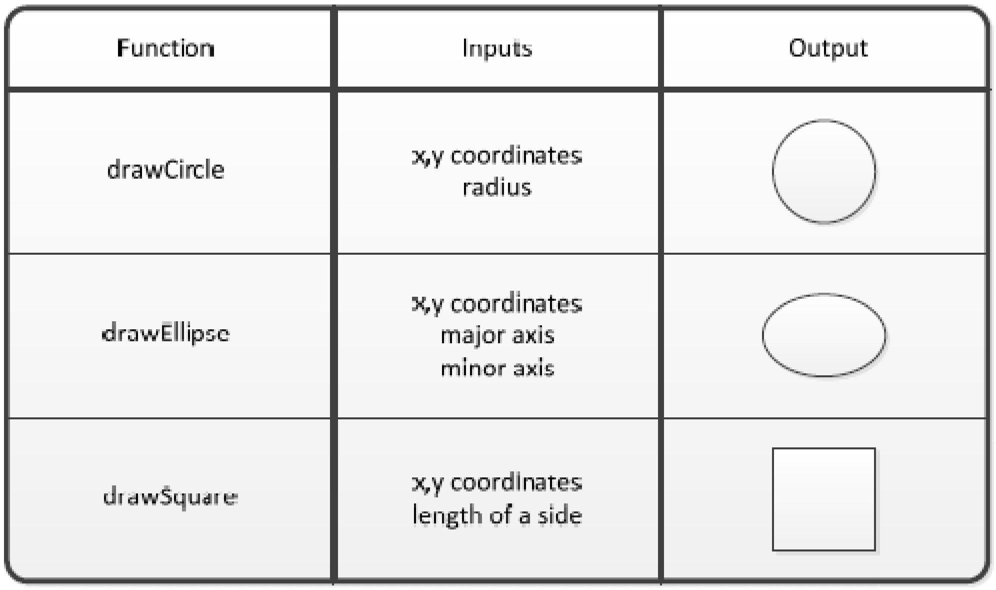
1. The Declaring Code and SSO of an API Embody Substantial Creativity in the Design of the API.

We understand that one of the factors in the fair use inquiry is "the nature of the copyrighted work," and that this involves an analysis of the creativity of the underlying copied work. In this case, we understand that it is undisputed that Google copied the "structure, sequence, and organization" (SSO) and declaring code of 37 Java API packages. The copied portions disclose what packages, classes, methods, and variables are contained in the Java libraries, their organization and relationships, how to use them, and their expected behavior.

There is substantial creativity involved in developing the declaring code and SSO of an API. This creativity is demonstrated by the fact that there are countless different ways that they can be expressed. For any given problem or use case, the declaring code and SSO can be structured and expressed in any number of ways, and that variety reflects the creative choices and subjective judgments of its author rather than the author's response to functional requirements.

There are iterative design choices at each level of an API, and the ultimate result is an expression of those choices. For example, a developer has to decide which methods, classes, packages, and other elements to include. In addition, the API designer must decide how these elements should be used, how they are supposed to behave, and how they relate to each other. Classes and packages can be rearranged, interfaces can be implemented in one API but not in another, and the SSO can take many shapes. Decisions must also be made regarding the declaring code. For example, decisions about the declaring code include how to name the methods, the selection and ordering of inputs and outputs, and what kinds of errors can be reported. Each step in the design process leaves room for the imagination and independent judgment of the author.

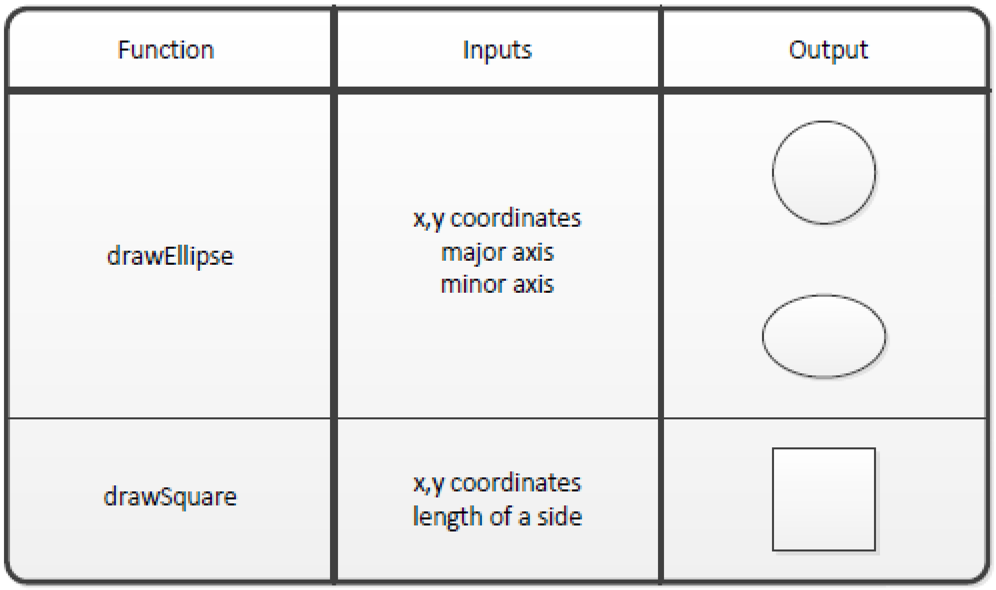
A simplified example is instructive. A Java API could be created to draw graphics on a screen, but it could be designed in many different ways. Figure 1 below shows one possibility, in which a "Polygon" class contains three separate methods for drawing each of three separate shapes: “drawCircle,” which takes as inputs the coordinates of the center of the circle and the length of the radius; “drawEllipse,” which takes the coordinates of the center, the length of the major axis, and the length of the minor axis; and “drawSquare,” which takes the coordinates of the center of the square and the length of a side.



Function	Inputs	Output
drawCircle	x,y coordinates radius	○
drawEllipse	x,y coordinates major axis minor axis	⬭
drawSquare	x,y coordinates length of a side	□

Figure 1.

However, these functions are not pre-determined, as the designer may instead choose to design the “Polygon” class differently. As shown in Figure 2 below, instead of a separate “drawCircle” method, the API may only include the “drawEllipse” function. This approach may be more difficult to use and implement, because the user must be sure to input equal values for the major and minor axes rather than simply the radius to draw a circle. However, it has the added advantage of greater flexibility and power, in that it allows a user to draw more complex shapes than a circle using the same method and to more simply modify and test code that uses the method.



Function	Inputs	Output
drawEllipse	x,y coordinates major axis minor axis	○ ⬭
drawSquare	x,y coordinates length of a side	□

Figure 2.

As a third alternative, instead of separate methods for drawing specific shapes, the author of the API may instead define within the "Polygon" class only a single method "drawPolygon" that takes as its inputs the coordinates of the center of the shape and a complex variable called "ShapeType." The "ShapeType" input in turn is defined in an entirely new class, and represents a complex data object that includes variables for the number of sides (*e.g.*, "0" for a circle or "4" for a square) and, for any given shape, the relevant size measurements (*e.g.*, the radius in the case of a circle or the length of the sides in the case of a square). While this method may be even more difficult to use and implement than those so far discussed, it is correspondingly more flexible and powerful because it can draw more complex shapes (*e.g.*, a pentagon).

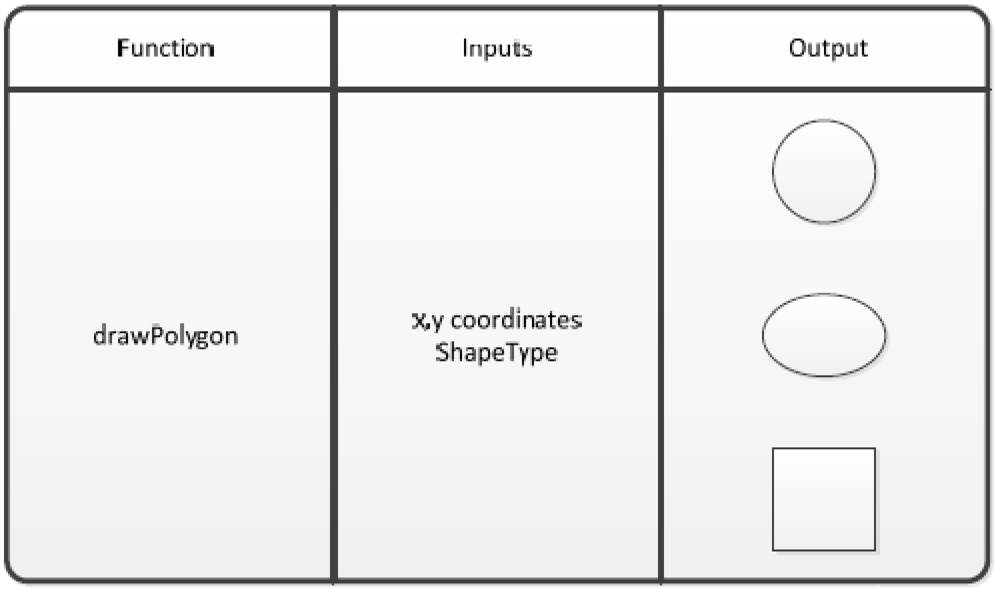


Figure 3.

Of course, these are only a few of the possibilities for a single method. There are many more ways that an author may design and structure the API to perform the same function. Moreover, the author must make similar decisions for each and every method and class in an API. For a complex API that may involve hundreds or thousands of methods and classes, these expressive choices are multiplied hundreds or thousands of times over, resulting in an almost countless number of ways an API design can be expressed. The many possible approaches to the simple circle-drawing example illustrates the myriad creative choices that an author must make, including balancing factors such as ease of use, flexibility, and difficulty of implementation.

That the result of this creative process is a functional product does not undermine the creativity involved in its design. The expression of an API is not dictated by function, but instead, is a direct reflection of the author's creativity. Accordingly, a court considering this factor of the fair use analysis should find that API packages, like those used in Java, are creative in nature.

2. The SSO and Declaring Code Are Significant Portions of the Creative Work of an API.

We understand that another factor in the fair use determination is the amount and substantiality of the portion used in relation to the copyrighted work as a whole, which includes a qualitative determination.

As discussed above, the SSO and declaring code of an API embodies a significant portion of the creativity involved in authoring an API. In some instances, the SSO and declaring code of an API can comprise most of the creative work of the API. As noted above, the selection and ordering of the packages, classes, and methods may have nearly uncountable options, as may the naming and selection of the declaring code. The implementing code, in contrast, may be relatively straight-forward or even limited by the confines of the creative decisions made in designing the SSO and declaring code. For such APIs, the creativity embodied in the SSO and declaring code exceeds that for the implementing code, and the SSO and declaring code comprise most of the creative work involved in the authorship of the API.

An example demonstrating how the SSO and declaring code can embody a relatively greater portion of the creative work of an API than the implementing code is instructive. Proceeding from the simplified example given above of an API containing a polygon-drawing class, the author of the class may choose to include one or more methods within that class to enable the drawing of one such polygon, a circle, in a number of ways. Two of the possible examples are shown above: The author could choose to include only a single method, "drawEllipse," with declaring code that relies on the user to supply major and minor axes of equal values. Alternatively, the author may include a separate, additional method, "drawCircle," with declaring code that specifies a single radius value as an input. The decision of whether to include one or both methods, as presented in authoring the SSO and declaring code, may involve creative choices, such as balancing ease-of-use against flexibility and power. Similar choices must be made for each of the methods that the author decides to include in the class, resulting in countless ways the design of the declaring code and SSO may be expressed.

By comparison, once those expressive design choices embodied in the declaring code and SSO have been made, the implementation of the chosen method(s) potentially may involve comparatively less creativity. The API author may find, for example, that writing the implementing code for drawing a circle is relatively straight-forward, and that there are only a finite number of ways to write the implementing code, given the limitations of the system and the inputs and outputs specified by the declaring code.

The SSO and declaring code also are significant portions of the expression of a software API because these are the portions that application developers see and use in writing programs. To learn how to write programs using the software API, for example, a developer could consult the specification of the API. The API specification identifies the libraries, packages, classes, and methods that are available for use by the developer, their declaring code, and their respective intended functions and relationships. The principal expression of the software API that is "exposed" to developers is the SSO and declaring code of the API. Developers need not ever view or even be aware of the implementing code in order to successfully learn and use the API. The declaring code and SSO instead connect the developers to the internal implementing code.

3. The Creativity of an API Is Not Substantively Changed by Substituting New Implementing Code for the API's Existing Implementing Code.

We understand that another fair use factor is "the purpose and character of the use," which has been interpreted by courts to include a determination of whether the use is transformative, that is, whether the use changes the expressive content, meaning, or message of the underlying work. We also understand that it is undisputed in this case that, in most instances, Google used different implementing code for the 37 Java API packages from which it copied the declaring code and SSO. But the implementing code for the copied packages in Android performed substantially the same function as the implementing code for the packages in Java. For example, the Android implementing code accepts the same parameters and generates the same return values and exceptions as the Java implementing code.

Indeed, we understand from the District Court that the implementing code written by Google replicated the exact same functionality as in Java. The SSO of the packages, classes, and methods, and their relationships to each other, remained the same and were not changed by Google's replacement of the implementing code. The declaring code similarly provided the same method specifications. The declaring code and SSO of the Java API packages that were copied in Android thus retained the same purpose, function, and meaning as they had in Java. The replacement of the implementing code in Android did not substantively change or add to the purpose or creative expression in the design embodied in the copied SSO and declaring code.

4. The Creative Expression of an API Is Not Substantively Changed When it is Used on a Different Form Factor.

We also understand that the determination of whether a work is transformative examines whether the work uses the copyrighted material for a different or distinct purpose. In this case, we understand that the District Court narrowly interpreted the facts of this case under the assumption that Android was developed for use on mobile devices, such as smartphones and tablets, and Java originally was developed for use on traditional computing devices, such as desktop and laptop computers. But even if the copied declaring code and SSO were implemented on a different form factor, that would not change the purpose, message, or meaning of the copied declaring code and SSO.

Traditional form factors, such as desktop and laptop computers, and mobile form factors which have achieved popularity more recently, such as smartphones and tablets, are all platforms for computing and running applications. From the standpoint of a software API, the form factors are all equivalent, because each is fundamentally a computer that can interpret and execute programs, regardless of how the computer is externally packaged. Indeed, the same processors and other hardware components may be used across the different form factors. For example, recent versions of the most popular operating system for full-sized PCs, Microsoft Windows, have been designed to work on tablets.[2] And some recent tablets and smartphones contain processors that also are found in full-sized PCs.[3] Recent trends like these demonstrate that mobile smartphones and tablets have

[2] *See, e,g.*, Michael Muchmore, "How to use Windows 10 Continuum," PC Magazine, Apr. 21, 2015, http://www.pcmag.com/article2/0,2817,2482299,00.asp.
[3] *See, e.g.*, Microsoft Surface Pro Specifications, https://www.microsoft.com/surface/en-us/support/surface-pro-specs (tablet containing Intel x64 architecture processor, which has been used in desktops and laptops).

increasingly converged with traditional laptop and desktop personal computers from a hardware capability perspective.

The purpose of, and creative expression for, the software API for an application programming platform is the same in all form factors: to describe the syntax, functions, variables, and data structures that a programmer can learn and use. And the role of the API in writing those programs is not altered by whether the program is executed on a computer that is small and mobile or large and relatively less mobile.

The adaptation of a software API for an application programming platform for use in another form factor does not change the purpose of the API. As such, Google did not transform the purpose or character of the portion of the Java API that it copied when it implemented it on smartphones and tablets, rather than on desktop and laptop computers.

5. There Is No Technical Need to Copy the Declaring Code and SSO of an API.

Based on our understanding, there was no need, from a technical or technological perspective, to copy the declaring code and SSO for a subset of an API, as Google did in this case.

If the purpose of copying an API is to enable the use of a programming language, then naturally the copied portion of the API would be expected to include only that which is necessary to use the programming language. That does not appear to be the case here.

In the District Court proceedings, we understand that the parties stipulated that 170 lines of more than 11,000 lines of code that Google copied were necessary to use the Java programming language. The remaining copied code was not necessary to use the Java programming language. Those additional lines of code are provided for the convenience of application developers. Developers can use these pre-written lines of code as building blocks to create their own applications, without having to re-create those building blocks every time they write a new application. The vast majority of the code copied by Google thus appears to have been taken from this collection of pre-written packages, which are not necessary to use the Java programming language.

We understand that in the trial on remand, Google in fact conceded that Android is not interoperable with Java. In the context of software and computer systems, "interoperability" refers to the ability of software designed for execution on one system to be executed on another. We understand that in this case it is undisputed that a program written for the Java platform cannot be compiled and executed on the Android platform. And conversely, a program written for the Android platform cannot be executed on the Java platform. Google's copying did not enable interoperability between the Android and Java platforms, and the fact that Google copied only certain Java packages and not others, as would be required for full interoperability, suggests to us as practitioners that its copying was not related to any technical needs or desire for interoperability.[4]

We understand that in the District Court proceedings on remand, Google has stated that it copied select portions of the Java APIs, not necessarily to achieve interoperability, but rather to encourage engineers, who were already familiar with Java and its APIs, to develop in Android. The District Court has called this copying that is necessary to prevent "cross-system babel." Order Denying Rule 50 Motions at 10 (Dkt. 1988). In other words, in lieu of claiming that the copying was necessary for cross-system *operation* (*i.e.*, interoperability), which Google acknowledged that its copying did not enable, the District Court instead stated that Google's copying was necessary for cross-system *consistency* (*i.e.*, to aid Java developers in learning to develop on Android more quickly).

Google's desire to make Android more attractive to Java developers is not borne of any technical necessity. It is common for application developers to learn multiple programming languages and to create programs for competing platforms. Developers can readily adapt to and learn new languages, particularly when the

[4] If Google wanted to achieve full interoperability, it could have taken a license and used all of the Java API. As set forth in the Federal Circuit's prior opinion in this case, Oracle offered three different types of licenses to Java. Based on the license selected, Google could have used Java. We understand that Google did not do so—it copied only select portions, and did so without a license.

language is conceptually similar to one that the developer already knows.[5] There is no cross-system consistency or cross-system "babel" problem as the District Court suggested. In this case, even though Google copied the declaring code and SSO of 37 Java packages, developers still had to learn the remainder of Android, which was not copied from Java, including the syntax, functions, variables, and data structures in Android that differ from those in Java. Java and Android remain different. Google's copying thus did not appear sufficient, let alone necessary, as a technical matter, to alleviate the sort of cross-system "babel" that the District Court noted.

C. Additional Considerations Relevant to Software Innovation and Development.

We understand that one of the factors in the determination of fair use is the effect of the use upon the potential market for the work. Accordingly, an additional consideration is that Oracle had provided means by which Google could have "fairly" used Oracle's Java code that would have preserved the market that Oracle had intended and created for Java as an "open source" project.

As noted above and discussed in the prior Federal Circuit opinion in this case, Oracle offered three different types of licenses to Java. *Oracle Am., Inc. v. Google Inc.,* 750 F.3d 1339, 1350 (Fed. Cir. 2014). Because Oracle had an

[5] For example, the Java programming language is one of several well-known object-oriented languages, such as "C++."

existing licensing scheme, there was an available avenue for Google to use the Java code fairly. Instead, Google copied Java APIs without transforming the work, enabling interoperability, or contributing back to the Java community, which Oracle's licenses sought to establish, encourage, and protect. This severely undermined Oracle's rights to control the market for Java and its derivative works.

Finally, because APIs are expressive works, permitting verbatim copying of APIs without legitimate justification will stymie innovation and development. Having robust, secure APIs is important. As in many fields, additional development and innovation of APIs requires programmers to bring their creativity and decision-making to bear. If these works are not protected, and can be verbatim copied without consequence and without compensating their original author, innovation will suffer, as programmers will be disincentivized from exercising their creativity. Similarly, permitting copying of APIs harms the rights of developers to control and create new versions of APIs, programming languages, and/or platforms, as they are unable to control how and when to develop new products and enter new markets.

While computer programming is highly disciplined in its structure, it nonetheless affords wide latitude in both implementation details and overall functionality. Similar to a composer, for example, the resulting work of a programmer reflects the vision and creativity of its maker. Indeed, it is common for a programmer to be identified simply by reading his or her source code.

Two primary factors that motivate a programmer to create useful source code are the enjoyment that comes from creating something that serves a purpose, and the promise of financial compensation, which sometimes can be quite significant. To remove the incentive of financial compensation would greatly reduce the motivation of those creative individuals who conceive and implement in solutions to important problems in software. Like most countries, the United States recognizes the incentive that copyright protection gives creative people and the benefit to the general public that ensues. This approach has given rise to a host of creative works that otherwise might never have seen the light of day.

IV.

CONCLUSION

We respectfully submit the foregoing technical considerations for the Court's attention in reviewing the fair use determination in this appeal.

Dated: February 17, 2017

Respecfully submitted,

/s/ *Jared Bobrow*

Jared Bobrow
Principal Attorney
WEIL GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
Telephone: (650) 802-3000

Counsel for Amici Curiae

CERTIFICATE OF COMPLIANCE

The undersigned certifies that this brief complies with the type-volume limitations of Fed. R. App. P. 32(a)(7)(B). This brief contains 5,643 words as calculated by the "Word Count" feature of Microsoft Word 2010, the word processing program used to create it.

The undersigned further certifies that this brief complies with the typeface requirements of Fed. R. App. P. 32(a)(5) and the type style requirements of Fed. R. App. P. 32(a)(6). This brief has been prepared in a proportionally spaced typeface using Microsoft Word 2010 in Times New Roman 14 point font.

Dated: February 17, 2017

/s/ *Jared Bobrow*
Jared Bobrow
Principal Attorney
WEIL GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
Telephone: (650) 802-3000

CERTIFICATE OF SERVICE

I hereby certify that on February 17, 2017, I filed or caused to be filed copies of the foregoing with the Clerk of the United States Court of Appeals for the Federal Circuit via the CM/ECF system and served or caused to be served a copy on all counsel of record by the CM/ECF system.

Dated: February 17, 2017

/s/ *Jared Bobrow*
Jared Bobrow
Principal Attorney
WEIL GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
Telephone: (650) 802-3000